USING AN ARTIFICIAL NEURAL NETWORK TO DETECT THE PRESENCE OF

IMAGE STEGANOGRAPHY

A Thesis

Presented to

The Graduate Faculty of The University of Akron

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Aron Chandrababu

May, 2009
USING AN ARTIFICIAL NEURAL NETWORK TO DETECT THE PRESENCE OF IMAGE STEGANOGRAPHY

Aron Chandrababu

Thesis

Approved: _________________________________
Advisor
Dr. Kathy J. Liszka

Accepted: _________________________________
Dean of the College
Dr. Chand Midha

Faculty Reader
Dr. Timothy W. O'Neil

Dean of the Graduate School
Dr. George R. Newkome

Faculty Reader
Dr. Tim Margush

Date

Department Chair
Dr. Wolfgang Pelz
ABSTRACT

The purpose of steganography is to hide the presence of a message. Modern day techniques embed pictures and text inside computer files. Steganalysis is a field devoted to detecting steganography in files and possibly extracting the hidden image or text. This thesis introduces a new idea for steganalysis, that of training an artificial neural network to identify images that have another image embedded in them. Two different types of artificial neural networks, a standard and a shortcut type, are trained for two different types of data sets. One data set contains images with and without hidden images embedded in them. The other data set is derived from calculating the luminance values of the files in the first data set. The experimental results show that the shortcut artificial neural network performs better than the standard trained network, but still does not yield good results. We compare these results to two well known steganalysis tools. To date, no steganalysis technique has shown much promise, but this is highly experimental research. Many questions remain unanswered, and this thesis forms the basis for future experiments with using an artificial neural network as a useful steganalysis tool.
ACKNOWLEDGEMENTS

This research is a continuation of the idea proposed by Jason Bowling of using the artificial neural network from FANN library. I am grateful to him for his contribution in the FANN library operations. Also, I sincerely thank Chuck Van Tilburg for configuring the cluster for the experiments. I appreciate my committee members, Dr. Tim O’Neil and Dr. Margush, for their valuable suggestions. My sincere gratitude goes to Dr. Kathy J. Liszka, my supervisor, for her ideas and guidance in this research. Thanks Dr. Liszka, you are the gem in this research work!
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
</tbody>
</table>

## CHAPTER

I. INTRODUCTION ................................................................. 1

II. STEGANOGRAPHY .......................................................... 5

  2.1 Carrier files ......................................................... 7

  2.2 Mechanics of Steganography ................................... 8

  2.3 Encoding Techniques ............................................. 9

  2.4 Steganography Tools .............................................. 13

III. STEGANALYSIS ......................................................... 18

  3.1 Steganalysis .......................................................... 18

  3.2 Common Steganalysis Techniques ............................ 18

  3.3 Steganalysis Using Luminance ............................... 20

  3.4 Steganalysis Tools ................................................ 22

  3.5 Future of Steganalysis .......................................... 23
IV. THE ARTIFICIAL NEURAL NETWORK ................................................................. 25

4.1 An Artificial Neuron Cell ................................................................................ 25

4.2 Fast Artificial Neural Network (FANN) ......................................................... 27

4.3 Artificial Neural Network Input Generation ................................................... 27

4.4 Artificial Neural Network Training .................................................................. 29

4.5 Artificial Neural Network Testing .................................................................. 33

V. THE IMAGE CORPUS ........................................................................................ 34

5.1 Image File Format ........................................................................................... 34

5.2 Color 24-bit RGB Image Corpus Generation .................................................. 36

5.3 Grayscale 8-bit Image Corpus Generation ...................................................... 38

5.4 Steg Image Corpus Generations ...................................................................... 39

5.5 FANN Input Generation .................................................................................. 41

VI. RESULTS .............................................................................................................. 44

6.1 Test Results .................................................................................................... 44

6.2 Experiments ..................................................................................................... 45

VII. CONCLUSION AND FUTURE WORK ........................................................... 52

REFERENCES ........................................................................................................... 54
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>51</td>
</tr>
</tbody>
</table>

| 6.1 StegSpy: Results when tested on color steg images | 51 |
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Hidden secret communications</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>A simple example of embedding a character in the bytes of image data</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>The Araxis tool shows differences in files with and without steganography</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>makeHide4PGP.sh: Hide4PGP execution command generation</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>doHide4PGP.sh: Auto-generated Hide4PGP batch commands</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Visual steg detection example</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Different Levels of Luminance in the image</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>Process of embedding the secret image on to the cover image</td>
<td>21</td>
</tr>
<tr>
<td>3.4</td>
<td>Change in sorted luminance value before and after embedding the secret</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Single Artificial Neuron Cell</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>Layers of Artificial Neural Network</td>
<td>26</td>
</tr>
<tr>
<td>4.3</td>
<td>Sample ‘training.data’ file</td>
<td>28</td>
</tr>
<tr>
<td>4.4</td>
<td>Shortcut Type FANN network</td>
<td>30</td>
</tr>
<tr>
<td>4.5</td>
<td>Output from train.c</td>
<td>32</td>
</tr>
<tr>
<td>5.1</td>
<td>Process steps for color images</td>
<td>35</td>
</tr>
<tr>
<td>5.2</td>
<td>Process steps for grayscale images</td>
<td>36</td>
</tr>
</tbody>
</table>
CHAPTER I
INTRODUCTION

Digital pictures are common in our world today, filling our email inboxes, and sitting on web sharing sites like Flickr [1] and Facebook [2]. However, consider the possibility that these harmless looking images could carry hidden secrets. Criminals, spies, and terrorists need secure means to communicate with one another without detection by law enforcement and intelligence officers. The intelligence community fears that steganography, a technique that hides information inside a carrier, will become a very effective tool. Modern day steganography embeds pictures and text inside computer files, known as cover files. These files can be simple email, text documents, images, audio files and video files. The cover files may then be exchanged through e-mails or simply hosted on a web server where another party can download them at their leisure.

Thanks to the overwhelming amount of information available on the Internet, one can easily find freely available steganography software. In 2007, an estimated 800 steganography programs were identified, many with user friendly interfaces [3]. The user typically does not need any technical, mathematical or programming knowledge to quickly embed secrets inside innocent looking digital images. With such widespread and easy access to these potentially danger tools, the Department of Homeland Security and other intelligence agencies are very concerned about the use of steganographic
techniques in the execution of terrorist and criminal activities. A quote from the *Federal Plan for Cyber Security and Information Assurance Research and Development* gives us a clear understanding of how serious this issue is: “International interest in R&D for steganographic technologies and their commercialization and application has exploded in recent years. These technologies pose a potential threat to U.S. national security. Because steganography secretly embeds additional, and nearly undetectable, information content in digital products, the potential for covert dissemination of malicious software, mobile code, or information is great.” [4]

The intended use of steganographic techniques is to embed information in such a way that it is not at all obvious to a casual observer that something is hidden, without prior knowledge of what is taking place and how it was woven into the cover file. Couple this with strong cryptographic algorithms, and it is virtually impossible for someone to detect the presence of steganography, recover the hidden information intact, and subsequently decrypt it. News media hype talks of terrorists exchanging critical information for attacks such as those on the World Trade Centers in 2001 [5]. However, other law enforcement agencies are concerned with the easy spread of hidden child pornography, or of the use of steganography to store stolen identity information [6].

**Contributions and Outline**

In this research, we present the following contribution to the field of steganalysis, the process of detecting the presence of steganography in cover files. Specifically, we are looking at a novel approach to detect the presence of hidden images inside other digital images that have been modified using the Least Significant Bit Alteration approach. We
have trained a series of artificial neural networks to identify those images with suspected embedded information. We do not attempt to identify the tools or the exact algorithm used to embed the hidden information, nor do we attempt to extract it, which at best is impractical with over 800 applications (i.e. 800 different possible signatures to identify). Without knowledge of the steganography technique applied, it is a formidable problem, indeed, to recover the hidden data. Our test steganography software is called Hide4PGP, a publicly available software package that provides a variety of options for embedding files into other files.

In Chapter two, we give a brief overview of the evolution of steganography, and then discuss the two basic mathematical and computational approaches for embedding information in digital media. We list the more commonly available steganographic tools and then describe Hide4PGP, the tool that we selected for use in this research.

In Chapter three, we discuss the basic approaches to steganalysis, a relatively new field in computer science. In our research, we use two steganalysis tools for comparison of the effectiveness of our artificial neural network approach to other research efforts. From our observation, the number of steganalysis tools is considerably smaller than the number of steganography tools. Few are freely available. We selected two for benchmarking and describe these tools in Chapter three.

An image corpus must be developed for this type of research. Chapter four discusses the steps in image preparation including collecting cover images, scaling them for compatible input into an artificial neural network, and embedding information with the selected steganographic tools.
Chapter five gives a basic background on the use of artificial networks and specifically, the FANN libraries used in this research. The approach selected for testing and training the network is also discussed.

In Chapter six, we present the results of our research, consisting of the output of many artificial neural networks trained with different data sets. We present the results of testing these networks on steg and non-steg images. These same test images are also analyzed using the steganalysis tools presented in Chapter three for comparison. Finally, conclusions and future work are provided in Chapter seven.
CHAPTER II
STEGANOGRAPHY

Steganography has historical roots dating back to around 440 B.C. The first account of what we call steganography appeared in the works of the Greek historian, Herodotus. He recorded an interesting tale about delivering tactical information over a large distance without being detected. A ruler named Histiaeus used a prisoner as his “steg” carrier by tattooing a message on the shaved head of the man. When his hair grew back, the prisoner was sent off to his destination, successfully, without the message being detected. Over the centuries, many other techniques have been used in warfare, in espionage, and to hide illicit affairs. In ancient China, messages were written on silk, dipped in wax, and swallowed. Many cultures have practiced steganography by writing messages using invisible inks. Concealment has taken many forms during the two World Wars, including arranging clock hands, encoding children’s report cards, reporting bogus baseball statistics, assigning meanings to popular song requests on the radio, and so forth. Indeed, the practice of steganography includes a rich and interesting history [7].

During World War I, a more modern technique called “microdots” emerged. It was verifiably used during both World War I and II. By the time the Berlin Wall was erected, the technology had advanced to where special cameras were used to photograph secret documents. The negatives were reduced in size and using special chemicals,
fastened to “dots” on newspapers on top of the letters “i” and “j”, or placed on the bottom side of a postal stamp. Interestingly, the idea of microdots was not initially embraced. It seemed too implausible, although the intelligence community had enough reason to believe microdots were being used that they assigned staff to research it. In fact, the first reported real one was found in 1941. Up to that time, they were known as “Zapp outfits” in reference to a German scientist who reportedly invented the technique and sold kits [8].

The common characteristic of modern digital hiding techniques and microdots is the stark fact that in both these techniques the hidden messages seem virtually impossible to detect. Today, we think of steganography as a digital technique for hiding an image or text inside another, innocuous-looking file. Most people equate cryptography with concealment, which is not entirely accurate. Encryption is meant to conceal the content of a communication; steganography conceals its very existence. When a message is simply encrypted into ciphertext, it is obvious to anyone that might receive or intercept the message that it is no longer readable plaintext. Knowing one is in possession of ciphertext, one can apply a cryptanalysis attack with the use of computer technology to identify and crack the encryption used to protect the message. If the message can first be encrypted, presumably with a strong algorithm, and its very presence hidden so that it would likely not be detected, then cryptanalysis techniques would never be applied and the message would more likely remain secure.

Combining steganography with cryptography can be accomplished in two basic ways. Secret key steganography requires a steg key be exchanged between communicating parties. This steg-key is used to embed the secret message in the cover
message and subsequently, to extract the hidden message at a later time. Public key steganography uses public key cryptography, where the sender encodes the hidden information with the public key before using the steganographic technique. To extract the information, the steganographic extraction tool is applied first, and then the private key is used for decoding. [7]

2.1 Carrier files

Virtually any type of digital file can be used as a carrier for steganography. The larger the file, the easier it is to conceal large amounts of information. Readily available and free tools exist that hide any type of digital information (viewed simply as a stream of binary digits) in images. Some use audio files, such as “.au”, “.wav”, and “.mp3”. Others use video files like the “.wmv” format, for example. Individual tools are limited in the number of formats they will support. Text file steganography uses entirely different techniques than image, audio or video formats. This type is also slightly more limited in the amount of data they can conceal.

Two popular text steganography tools are SNOW [9] and Spam Mimic [10]. These two tools use “white space” steganography, meaning they hide information by adding tabs and blanks at the end of lines in the text. Snow is a DOS command line tool that embeds text inside other text files. The Spam Mimic web site has an online tool that generates a meaningless spam message for you with your hidden data inside. The web site tool determines the length of the generated spam text. It also provides the option to use encryption on your data before embedding it into the generated text. Once done, the
user simply copies the message into an email. The email, once received, can be copied back into the Spam Mimic website, and the hidden text extracted. If encryption was used, the key must also be provided. The latest twist lets a user create a fake PGP header with a hidden message. The most important aspect, regardless of the technique used, is to use a carrier file that, once altered to carry the hidden message, shows no evidence of alteration.

A particularly insidious form of steganography uses packet headers used in the transmission of data. This is called *protocol steganography* [11]. For example, an IP packet header can be modified by adding hidden data in unused fields. The header, along with its normal payload, is sent across the Internet and the data extracted from the IP header at the point of reception.

In this research, we concentrate solely on image steganography which is believed to be the most widely used because of its higher capacity of carrying hidden data and because it is very difficult to differentiate an image with steganography from a normal digital image.

### 2.2 Mechanics of Steganography

Here, we present the basic ideas involved in hiding a secret image in the image carrier file. The first step is to select a cover image. It would seem most appropriate to select an innocent looking image. The next step is to select, install, and run a steganographic tool to embed the secret in the cover image. Once embedded, we refer to this file as a *steg file* which can be sent to a receiver. Once the steg file is received, the
intended recipient should know how to reverse the process. The same steganography tool is used to extract the hidden message from the steg file. Figure 2.1 depicts the flow of the secret communication.

![Figure 2.1 Hidden secret communications](image)

2.3 Encoding Techniques

The number of possible encoding techniques is limited only by the imagination. Here, we present the commonly used tools that are found on the Internet [12].

2.3.1 Attachment

The secret message, or data, is appended to the carrier file. In this case, the size of the carrier file will be changed. The white space techniques used by SNOW and Spam
Mimic fall into this category. This technique is the easiest to detect, compared to other techniques.

2.3.2 Headers

Here, unused fields or positions in the file header are used to store the information. This is quick and simple, but has as a disadvantage the small amount of information that can be hidden. It can also be easily detected by analyzing the header for known tags versus unidentifiable information.

2.3.3 Dispersal Algorithms – Least Significant Bit (LSB) Alteration

Digital images are stored by using pixel values which are represented by bits. In this technique, an algorithm is used to disperse the bits of the secret data inside the carrier file by modifying the least significant bits of an image’s RGB values. To the naked eye, the difference in color resulting from a change of the least significant bit is undetectable. In general, if the three least significant bits are replaced by steg data then one may start noticing differences in the color scheme of an image. When the steg image is compared with the original image, the modified pixels can be noted. Figure 2.2 shows a simple example of how data can be hidden in the LSBs of image data. In this example, each 8 bit grayscale pixel is represented by one byte and the LSB of each of these bytes are modified to store the hidden information.
Figure 2.2 A simple example of embedding a character in the bytes of image data

Figure 2.3 shows the highlighted pixels in a carrier image that is modified after embedding a secret message. We used a tool called Araxis to see the modified pixel positions of a generated steg file [13]. The highlighted grey dots in the right hand side of Figure 2.3 represent the pixels changed when a steg image is embedded into the original image. There are no visual changes between the steg image and the original image. The Araxis tool highlights pixels that are changed by comparing with the original image. For this example, the Hide4PGP tool is used to embed a secret image in the carrier image. 
This type of technique is very hard to detect if programmed well and if the user carefully selects a carrier file that is large enough to hold the steg data.

![Figure 2.3 The Araxis tool shows differences in files with and without steganography.](image)

2.3.4 Dispersal Algorithms – Discrete Cosine Transform Coefficient Alteration

This is a variation of the simple LSB technique. In this case, a discrete cosine transform (DCT) is used to transform 8 x 8 pixel blocks of the image into 64 DCT coefficients. This technique is used for files stored in the JPEG image format. The redundant bits selected to embed the hidden data are taken from the least–significant bits of the quantized DCT coefficients [6]. The modification on a single DCT coefficient
affects all 64 image pixel blocks. Thus the smoothening of the pixel alteration is virtually impossible for human visual detection.

2.4 Steganography Tools

For those that lean towards the dark side of computing, there are many tools available just a keystroke away. While it might seem reasonable to simply track and restrict or remove access to steganography tools, it is implausible. Due to legal restrictions between different countries, the sites that host these tools cannot be banned. Some steganography sites require registration, which can be spoofed. Others don’t bother and, in fact, encourage anonymity. In other words, anyone is free to download and use the software.

In this research, we used one steganographic tool to create our steg corpus in training the artificial neural network. A corpus is the set of images collected for our research. The selected tool, described in the following sections, is Hide4PGP.

2.4.1 Hide4PGP

Hide4PGP [16] is a free steganographic tool widely available on the Internet. Hide4PGP uses an LSB alteration technique on 8 bit and 24 bit bmp cover images. The default hiding bit is the LSB of an 8 bit image and the fourth LSB of a 24 bit image.
We now present the steps to embed and extract data using the Hide4PGP tool. For embedding the secret data inside the cover image following command is used.

\textit{Hide4PGP picture.bmp secret.jpg}

Here, ‘secret.jpg’ is the data file that is embedded in the cover image ‘picture.bmp’. The ‘-x’ option is used to extract the secret data from ‘picture.bmp’ and store it in the file ‘s.jpg’, which resides in the same folder as that of the cover images.

\textit{Hide4PGP -x picture.bmp s.jpg}

As an initial test to determine whether the Hide4PGP tool works as advertised, we embedded cover images with secret images and successfully extracted the secret images back from the cover images.

In order to streamline the process of generating thousands of steg images, we created a large number of Hide4PGP commands with the help of a UNIX Shell script. Figure 2.4 shows the UNIX script that generates the Hide4PGP commands to embed the secret data in all files in the specified folder. While executing this script, two lists are produced; One list of plain images and other of secret images. Both the list should have same number. Then each list names is matched and then a Hide4PGP execution command is generated. These auto-generated embedding commands are written to the file “doHide4PGP.sh”.

14
Figure 2.4 makeHide4PGP.sh: Hide4PGP execution command generation

```bash
#!/bin/sh

# Generate the Hide4PGP commands for multiple image steganography

echo '#!/bin/sh' > doHide4PGP.sh
echo '' >> doHide4PGP.sh

ls StegTools/Hide4PGP/gl/Org_steg/1.1
ls StegTools/Hide4PGP/gl/secret/1.2
paste 1.1 1.2 >11

cat 11 | awk '{print "hide4pgp ./StegTools/Hide4PGP/gl/Org_steg/"$1 " ./StegTools/Hide4PGP/gl/secret/"$2""}’ > doHide4PGP.sh

chmod u+x doHide4PGP.sh
```

Figure 2.5 doHide4PGP.sh: Auto-generated Hide4PGP batch commands

Figure 2.5 shows the batch of Hide4PGP commands generated from “makeHide4PGP.sh”. On execution of the “doHide4PGP.sh” script, all the images in the specified folder will be transformed to steganographic images using the secret images.
2.4.2 Other Steganographic Tools

Although we only used Hide4PGP in our research to build the corpus, there are many other popular steganography tools freely available. Some other tools that we considered are briefly described.

- **wbStego4open**: This is an open source application that works in both the Windows and Linux platforms. It supports Windows bitmaps, ASCII or ANSI text files, HTML files, and Adobe PDF files for the carrier files. This application is very effective for embedding copyright information without noticeably modifying the carrier file. wbStego4open does not require registration[18].

- **Invisible Secrets**: This is proprietary software marketed to hide data in image or sound files. It also includes the AES encryption algorithm for extra protection. There is a password management feature that creates and stores the secure passwords used in creating the steg files, making it user friendly [19].

- **S-Tools**: In this steganographic tool, bmp, gif, and wav files are used as the carrier files. Drag and drop features make it very easy and convenient to create steg files. It also supports its own brand of encryption [20].

- **Hiderman**: This tool is proprietary and therefore, does not make its algorithm public. A 30-day trial version is available so that the user can experiment with
the user-friendly GUI. The cost for unlimited use is only $35.00. However, all of the available documentation appears to be in French. As with the others, it employs some form of encryption, but does not indicate which algorithm [21].

We have covered the basics of steganography and a sampling of the techniques and tools available. In the next Chapter, we discuss basic steganalysis. We describe one technique and two tools used in this research for comparison to our artificial neural network results.
3.1 Steganalysis

The purpose of steganalysis is to identify the presence of an embedded message, not to actually extract the message. The efficient detection of steganography is a challenging task. Much research has been conducted to develop a better method of steganalysis detection. In the following section, we will discuss some of the steganalysis techniques and tools that are currently available.

3.2 Common Steganalysis Techniques

In general, the redundant bits of the cover file are used to hide the secret message. However, the process of hiding may leave behind traceable signatures in the cover image. A distortion in the carrier image properties enables an eavesdropper to detect the presence of a hidden message. For some of these techniques, we have to have original cover file along with the steg file to do the comparison. Some common steganalysis methods [22] include:

Visual Detection: This can be used for pictures in JPEG, BMP and other image formats. When the carrier picture file is embedded with secret data, distortion may appear
in the original image. This distortion can be identified with the naked eye. Figure 3.1 shows an example of visual steg detection.

![Original Image](image1.png) ![Steg Image](image2.png)

Figure 3.1 Visual steg detection example

**Audible Detection:** In this case, a distortion in WAV and MPEG files can be taken into account to identify the presence of secret embedding.

**Structural Detection:** This is done by comparing the steg file’s properties and contents with those of the original file. The properties that are likely to be changed include:

- A difference in file sizes
- A difference in dates or times
- A modification of the content body
- A change in the checksum values
Statistical Detection: This method works by analyzing the changes in pixel patterns, in particular, the least significant bits. This type of detection is also known as histogram analysis [22]. $\chi^2$ steganalysis is a common statistical method used for files in the JPEG format [6]. As we have discussed in the previous Chapter, the least significant bits of the quantized DCT coefficients can be used as the redundant bits for embedding a message. Thus, $\chi^2$ steganalysis makes use of the difference in the DCT coefficient frequencies for detecting the presence of hidden data [6].

Signature tracing: This method relies on the signatures that are left behind in the carrier files during embedding. We will discuss more about this in the next section.

3.3 Steganalysis Using Luminance

Luminance is defined as amount of brightness that is emitted by a pixel or area on an image. For example, bright red and dark red have the same chrominance value but a different luminance [36]. Figure 3.2 shows the different levels of luminance for the same picture [37]. The image on the left in the figure contains the least average luminance value and the image on the right has the highest average luminance value.

Figure 3.2 Different Levels of Luminance in the image
In our experiments, we have found the luminance values of the image changed after embedding a secret image into a clean image. Figure 3.3 shows the process we used to convert a clean cover image into a steg image.

![Figure 3.3 Process of embedding the secret image on to the cover image](image)

One cannot truly see a difference with the naked eye. For our experiments, we calculated the pixel luminance of both the images using the following formula [38]:

\[
\text{Luminance} = (0.299) \text{ Red} + (0.587) \text{ Green} + (0.114) \text{ Blue}
\]

After luminance is calculated for each pixel, the values are sorted. We have also conducted another set of experiment without sorting the luminance values. To check whether there is any change in luminance, we used Araxis Merge software to compare the cover image before and after embedding. Figure 3.4 shows the change in luminance value of the cover image. We conjecture that this pattern of luminance value change may be useful in detecting the presence of steganography.
3.4 Steganalysis Tools

A number of steganalysis tools are freely available on the Internet. However, from our observation, there is a much smaller set of tools, whether free or for a fee, than the number of steg tools available. We selected two free software tools, StegSpy and StegSecret to use as a comparison to the results generated by our artificial neural network experiments.

**StegSpy:** This tool was developed in response to the terrorist attacks in the United States that occurred on September 11, 2001[23]. StegSpy is a Visual Basic program based on the signature identification technique. It searches for a steg signature and tries to identify the program used to embed an image. StegSpy claims to detect different types of
steganography. It also shows the position in the file where a steg image is most likely to be stored.

**StegSecret**: This software is another steganalysis tool written in Java for platform independence. It claims to detect attachment type, LSB type and DCT type steganography. It focuses on image, audio and video carrier file types. This project is still being enhanced by incorporating more steganalysis algorithms[24].

**StegAlyzerSS**: SARC (Steganography Analysis and Research Center) has its own steganalysis tool which is called StegAlyzerSS. It is a digital forensic analysis tool with the capability of detecting steg files and then automating an extraction algorithm which will try to extract the hidden information. Its main technique is to scan the digital files to find the known signatures or hexadecimal patterns of specific steg algorithms [25]. Since it is proprietary software, many of the inner details are not revealed.

3.5 Future of Steganalysis

Detecting the presence of steganography in an image is a difficult goal to accomplish. Statistical steganalysis techniques require a lot of mathematical computation for each unique type of steg detection. As new stenographic techniques are developed, the statistical computations will need to be customized to be effective. It will be harder if we have to apply all statistical techniques in order to detect a single kind of steganography. Thus, in this research, the focus is on a general method to detect only the presence of a steganographic image made from one steganography program, Hide4PGP, with two variations on embedding an image. This will be discussed further in Chapter 5. We hope to adapt our technique of using an artificial neural network to other
steganographic algorithms. The next Chapter shows how an artificial neural network is trained for steg image detection.
CHAPTER IV

THE ARTIFICIAL NEURAL NETWORK

An artificial neural network (ANN), also called simply a "neural network" (NN), is a computational model based on biological neural networks. It consists of an interconnected collection of artificial neurons. An ANN is an adaptive system that changes its structure based on information that flows through the artificial network during a learning phase.

Like a biological brain, an ANN is designed to learn from sample situations. The brain consists of billions of neurons and their inter-connected networks. The brain learns by sending a pulse or signal to neurons. In fact, our knowledge of the brain is very limited. However, the basic concept of learning is mimicked in designing an ANN. The ANN is based on the principle of learning by example [32]. We first describe the structure of the artificial neuron.

4.1 An Artificial Neuron Cell

An artificial neuron is composed of two functional units. The adder computes the sum of the products of the weight coefficients and the input signals. The activation function produces the output signal $y$ from the adder output signal $e$. Signal $y$ becomes
the output for the neuron [31]. Figure 4.1 shows the structure of a single artificial neuron cell [31].

![Figure 4.1 Single Artificial Neuron Cell](image1)

The simple basic neural network consists of three different layers: the input layer, the hidden layer and the output layer. Figure 4.2 shows how the layers are fully connected. Each node in one layer is connected with every other node in the following

![Figure 4.2 Layers of Artificial Neural Network](image2)
layer. Every connection has a weight associated with it. The initial weights may vary with
different applications, but have a significant role in the learning phase.

The network output signal $y$ is compared with the desired output value $z$, which is
found from the training data set. The error signal $\delta$ is the difference of $z$ and $y$. This error
signal $\delta$ is propagated back to all neurons. By doing this, the weight coefficient is
adjusted so as to make the output closer to the desired output value [31].

4.2 Fast Artificial Neural Network (FANN)

The fast artificial neural network (FANN) [33] was selected for our research as
the neural network library in order to verify our hypothesis that an ANN can be trained to
detect steganography in images. It is a free open source library that generates a
multilayered ANN in the C language with additional support for implementing both fully
connected and sparsely connected networks. The FANN library can be used with C,
C++, PHP, Python, and Delphi. In this research, we are calling the FANN libraries from
C programs. The back-propagation strategy that we have discussed in the previous
section is used in training the FANN.

4.3 Artificial Neural Network Input Generation

First, we generate an input file which is in a format that the FANN can read. The
input file creation script takes two parameters, a directory with the images and either 1 or
-1 depending on the expected output for the particular image. After listing the images in ‘convert_2_train.sh’, the script can be used to convert various images into the FANN input format.

The ‘training.data’ file that contains the preprocessed input is shown in Figure 4.3. The last step of the training phase is to train the artificial neural network with our preprocessed training data.

![Image](image.png)

Figure 4.3 Sample ‘training.data’ file

The auto-generated ‘training.data’ file contains many line pairs and a header line. The first line of each pair contains 22,500 scaled pixel values from the image. A second line is paired with the image data that represents the training value. For our experiment, 1 is used to indicate steg and -1, non-steg. The header should conform to the FANN input format:

```
<# input images> <# inputs signal> < # output nodes >
```
The file data shown in Figure 4.3, contains data for 500 images, 22500 input signals and one output signal.

4.4 The Artificial Neural Network Training

The ANN is created using the FANN libraries commands mentioned in the FANN document [33]. The C programs are written using basic FANN functions necessary to create, train and run the FANN. The original version of C program code we started with was written by Jason Bowling [34]. It was developed for the identification of spam images.

We start with the `fann_create_standard` library function to create what is referred to as a fully connected back-propagation neural network. This is the basis of our starting experiments. Our attempts were dismal, showing no ability to train the FANN for our purposes. This led us to delve further and experiment with a modified network generated using the `fann_create_shortcut` library function. This function creates a standard fully connected back-propagation network but with extra “shortcut connections”. Shortcut connections directly skip some of the hidden layers. It also includes an additional direct connection from the input layer to the output layer [29]. In the course of our experiments, we found that this type of shortcut FANN showed a measurable increase in accuracy compared to the standard FANN. The number of shortcut connections is one of the factors that can influence the learning capability like the number of hidden nodes and the number of hidden layers. Each shortcut connection will also get trained using back propagation. The FANN works using the supervised learning paradigm, in other words,
the human led teaching approach. That is, the FANN is fed with training images, explicitly marked whether the image is steg or non-steg. The basic idea behind this approach is that if the ANN is given enough reliable learning information, then the network acquires the capacity to distinguish steg images by itself.

For our research, the FANN is created with 22,500 inputs two hidden layers with 50 neurons each, and one node for output. Figure 4.4 shows the shortcut FANN. The scaled values for each pixel are given to the input node. The single node output layer determines steg or non-steg from the interval between the values -1 and 1. The function `fann_read_train_from_file` reads and loads the data file created in image2fann.cpp. The `fann_set_activation_steepness_hidden` and `fann_set_activation_steepness_output` values
are set to 1 for the hidden and output layers respectively. In the FANN libraries, the steepness determines behavior of the activation function in each neuron (or node). Steepness value can be in the range of 0 to 1. A steepness value of 1 is used in this application in order to classify images as steg or non-steg.

The initial weights of the nodes are set with the function `fann_init_weights` using Widrow and Nguyen’s algorithm [35]. An even distribution of weights across each input node’s active region is used in our research.

The network learns by continually adjusting each node’s weights so that the output of the ANN matches the expected output value given in the training file. An epoch is defined as one cycle of operation where the weights of each node are adjusted to match the output in the training file. The function, `fann_set_learning_rate` is used to determine how aggressively we want the learning algorithm to behave. The images are trained to a desired error by using the function `fann_train_on_data`. This is accomplished by stepping down the training rate at preset intervals to avoid the possibility of oscillation.
The function \textit{fann\_run} takes a single image as the input to the network and outputs a single value between -1 and 1. Figure 4.5 shows the sequential procedure. The summary of the training results are displayed. Finally, the allocated memory is freed and the ANN saved by using the \textit{fann\_save} and \textit{fann\_destroy} functions.

As mentioned previously, back propagation is used to train the neural network. Each pixel value of the digital image can be fed to the input layer with the single neuron at the output layer, producing the results. There are only two kinds of results; positive or negative steganographic image detection which is indicated by values between -1 and 1 [30].
During the training phase, one set of steganographic images and another of clear images are given to the neural network so that the weight coefficients will be adjusted to recognize the patterns of the steganographic images.

4.5 The Artificial Neural Network Testing

The network is now tested against a collection of known images. During the testing phase, the weight coefficients will not be adjusted. The function \texttt{fann\_test} is used in the testing phase. The network is tested with both trained and untrained images to observe how well it has learned to identify steg and non-steg images. In the next Chapter we discuss how to construct a corpus of training images for a neural net.
CHAPTER V
THE IMAGE CORPUS

Steganography can be of different types as is mentioned in Chapter two. It can be stored in text files, digital images or even in audio files. Because digital images are so prevalent on the World Wide Web, we focus this initial study solely on image-based steganography. In order to ensure a clean corpus, we chose to use only images we generated ourselves with personal digital cameras. There are no statistics on how much “steg in the wild” exists. Viewpoints vary widely from none to an alarming amount. We decided not to risk downloading images that had even a slight potential of having steg already embedded in it. By doing this, we guarantee what we refer to as a “clean corpus” as our starting point.

5.1 Image File Format

Image file formats are standards used in digital images. In our research, BMP, JPEG, PNG and GIF formatted images are considered in building our image corpus. The majority were JPEG and BMP. We converted the JPEG, PNG and GIF images to BMP, the format we chose for our training standard.

The BMP file format, also known as DIB (Device Independent Bitmap) format, is used to store bitmap digital images, mainly in Microsoft operating systems. The
drawback of this format is that the file size is usually large compared to other formats. JPEG (Joint Photographic Experts Group) is an image compression format that has 8 bits per color i.e. a total of 24 bits [27]. It produces a relatively small size image.

Our experiments are conducted on two pairs of 150 * 150 BMP image corpuses. The first corpus is composed of 24-bit RGB color images and the other contains 8-bit grayscale images. Figures 5.1 and 5.2 show the process steps for color and grayscale images respectively.

![Diagram](image.png)

**Figure 5.1 Process steps for color images**
5.2 Color 24-bit RGB Image Corpus Generation

The original image corpus is a collection of personal images. These digital images vary in their file formats, sizes and dimensions. Since we decided to keep the BMP file format as the standard input format for the steganographic tool, all the images are converted to BMP using ImageMagick [28]. The reason for choosing BMP is that commonly used steganographic tools accept BMP format as the cover image. Hide4PGP, our chosen experimental tool, uses the BMP format image as the carrier.

After the image conversion is done, all images are 150 * 150 pixel BMP format images with the name extension “_150.bmp”. The corpus generation UNIX script is
shown in Figure 5.3. The ‘convert’ function of ImageMagick is used to convert all the
types of image formats into BMP format.

```
#!/bin/sh
# Create BMP file format images from any other image format
echo "#!/bin/sh" > c2bmp.sh
echo "" >> c2bmp.sh
for f in `ls $1`
do
echo "convert ./in2bmp/"$f" -resize 150x150! ./out2bmp/"$f"_150.bmp" >> c2bmp.sh
done
chmod u+x c2bmp.sh
```

Figure 5.3 All2bmp.sh: Preprocessing Image

```
#!/bin/sh
convert ./in2bmp/010-singapore_small.jpg -resize 150x150! ./out2bmp/010-singapore_small.jpg_150.bmp
convert ./in2bmp/011-singapore_small.jpg -resize 150x150! ./out2bmp/011-singapore_small.jpg_150.bmp
convert ./in2bmp/012-singapore_small.jpg -resize 150x150! ./out2bmp/012-singapore_small.jpg_150.bmp
convert ./in2bmp/013-singapore_small.jpg -resize 150x150! ./out2bmp/013-singapore_small.jpg_150.bmp
convert ./in2bmp/014-singapore_small.jpg -resize 150x150! ./out2bmp/014-singapore_small.jpg_150.bmp
convert ./in2bmp/015-singapore_small.jpg -resize 150x150! ./out2bmp/015-singapore_small.jpg_150.bmp
```

Figure 5.4 Generated preprocessing ‘convert’ script

This generated script converts all images in the ‘in2bmp’ folder and outputs the
formatted BMP file to the ‘out2bmp’ folder. At that point, the images from the
‘out2bmp’ folder are considered “clean” 24-bit 150*150 images for our experiments.
Each original image is converted to 150*150 RGB images using the ImageMagick tool as
shown in Figure 5.5.
5.3 Grayscale 8-bit Image Corpus Generation

A grayscale image contains a range of gray color pixels that vary from pure white to jet black. Each pixel is represented by a decimal value in the range of 0 to 255, that is, by 8 bits ranging from binary 00000000 to 11111111. Because these images do not use color, (the red, green, and blue used for 24 bit color) the 8 bit binary representation is called an “8 bit grayscale image”.

The color 24-bit 150*150 RGB images that are generated from the above conversion are used to make the 8-bit grayscale images. Figure 5.6 shows how these 8-bit grayscale images are generated.
5.4 Steg Image Corpus Generations

After both the clean image corpuses are generated, they are used to build a set of corresponding steg corpuses. Using Hide4PGP, secret images are embedded in each of the clean images. Figure 5.7 shows the steg image generation for 24-bit RGB color images. Figure 5.8 shows the steg image generation for 8-bit grayscale images.
Figure 5.7 Generation of 24-bit RGB Steg Image

Figure 5.8 Generation of 8-bit Grayscale Steg Image
5.5 FANN Input Generation

At this point, we have created four image corpuses. The clean 24-bit color RGB corpus and steg 24-bit color RGB corpus are combined and used for training and testing luminance. The clean 8-bit grayscale corpus and steg 8-bit grayscale corpus are ready to generate a FANN input file.

Case 1: FANN input generation using 24-bit color RGB corpuses

- Both the clean RGB 150*150 image and steg 150*150 RGB image consist of 
  
  \[(150\times150)\times3 = (22,500 \times 3)\]

  pixel values that range from 0 to 255.

- Calculate the luminance from the red (R), blue (B) and green (G) components using a program written in Java.

- Sort the 22,500 luminance values.

- Divide the sorted luminance values by 256 to scale them between 0 and 1 in order to be given to the neural network.

- The scaled values are written to the ‘training.data’ file with a format that will be accepted by the FANN.

Case 2: FANN input generation using 8-bit grayscale corpuses
• Both the clean grayscale image and steg grayscale images consist of $150 \times 150 = 22,500$ pixel values that range from 0 to 255.

• Read this file byte by byte and divide each pixel value by 256 to scale it between 0 and 1 in order to be given to the neural network. This is done by a program written in C++.

• The scaled values are written to the ‘training.data’ file in a format that will be accepted by the FANN.

Figure 5.9 FANN Input generation
Figure 5.9 shows the process of generation of the ‘training.data’ file. This training.data file is used as the input into the FANN for training. In the same way, test.data also can be generated for testing the FANN.

We have collected clean images, created a standard set of BMP format images, and created a set of corpuses for experiments to be run on luminance values and 8-bit grayscale values. The next Chapter summarizes our experiments.
CHAPTER VI
RESULTS

6.1 Test Results

We start our FANN experiments with a clean corpus of steg and non-steg images that are in a format that conforms to the required FANN input format. The ‘test.data’ file is generated by the ANN preprocessor from the images that are tested for the presence of steganography. This file has the same syntax as that of the ‘training.data’ file (i.e., a header at the beginning and the image line pair) but in this case the expected output for each image is zero, because we are not telling the ANN whether the images are steg or non-steg.

In the testing phase, the ‘test.data’ file is given as the ANN input. Figure 6.1 shows a sample output file. Each line represents the outcome for one image specified in the ‘test.data’ file. The first value is the single output value from the ANN which will be between 1 and -1. The second value is what we have given in the ‘test.data’ file for that particular image. Since we are testing the existence of steganography neutrally, we set this value to zero. The difference of these values is displayed as the last value. Each tested image has its results displayed on a single line in the output.
6.2 Experiments

We started our research by performing steganographic detection experiments with the two models described in Chapter four. From our experiments, we found that the “standard” fully connected back propagation model was not producing any meaningful results. We then tested our corpus using the standard fully connected back-propagation network but with shortcut connections which have a very particular network pattern. This ANN started giving better results compared to standard type ANN. We have also changed order in which the images were fed to the network in order to see if it produced different results. Where we refer to “mix” in the results, we interleaved steg and non-steg images as they were fed to the network for training. Where we refer to “non-mix” in the results, we fed all of the non-steg images followed by all of the steg images during the training process.

After the training phase, the ANN is tested with “seen” images (those that were part of the original training set) and then further tested with as yet, unseen images. This
let us look at initial data to study the FANN recognition capabilities. For each of our experiments, we trained the FANN with 250 Hide4PGP steg images and 250 non-steg images. In all cases, the ANN contained two hidden layers with 50 neurons each. The networks were trained with 11000 epochs. After the training was complete, tests were conducted with 50 StegHide steg images and 50 non-steg images.

In our experiments, we created two sets of trained FANNs. One FANN was trained with the luminance values calculated from the 24-bit RGB images. The other FANN was trained with the 8-bit grayscale image data.

6.2.1 Test results on Grayscale Images

Figure 6.2 and Figure 6.3 show the test results for grayscale images.

<table>
<thead>
<tr>
<th></th>
<th>Test: 50 Steg &amp; 50 non-steg = 100 total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CorrectSteg IncorrectSteg CorrectNonSteg IncorrectNonSteg Don't Know</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>E1_Gray_Mix_Standard Test Type:(Seen Images)</td>
<td>1 0 0 0 0</td>
</tr>
<tr>
<td></td>
<td>Test Type:(Unseen Images) Test: 50 Steg &amp; 50 non-steg = 100 total</td>
</tr>
<tr>
<td></td>
<td>CorrectSteg IncorrectSteg CorrectNonSteg IncorrectNonSteg Don't Know</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td></td>
<td>0 9 9 0 82</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Test: 50 Steg &amp; 50 non-steg = 100 total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CorrectSteg IncorrectSteg CorrectNonSteg IncorrectNonSteg Don't Know</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>E2_Gray_nonMix_Standard Test Type:(Seen Images)</td>
<td>0 0 0 0 100</td>
</tr>
<tr>
<td></td>
<td>Test Type:(Unseen Images) Test: 50 Steg &amp; 50 non-steg = 100 total</td>
</tr>
<tr>
<td></td>
<td>CorrectSteg IncorrectSteg CorrectNonSteg IncorrectNonSteg Don't Know</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td></td>
<td>0 0 0 0 100</td>
</tr>
</tbody>
</table>

Figure 6.2 Experiments with grayscale image on standard FANN
From our experiments with grayscale images, the shortcut type FANN was classifying the images better than the standard type FANN. In the case of the standard type FANN, most of the images were classified into the Don’t Know section.

6.2.2 Test results on Color Images

These training results are for the color images using the calculated luminance factor.
Figure 6.4 and Figure 6.5 show the test results on color images after training the FANN with sorted luminance values. From our experiments with color images, the shortcut type FANN was classifying images with significantly different results than the standard type FANN. In case of the standard type FANN, most of the images were classified into the Don’t Know section.
6.2.2 Without Sorting Luminance Value

<table>
<thead>
<tr>
<th>E9_Color_Mix_Standard_NonSorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Type: (Seen Images)</td>
</tr>
<tr>
<td>CorrectSteg IncorrectSteg CorrectNonSteg IncorrectNonSteg Don't Know</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>Test Type: (Unseen Images)</td>
</tr>
<tr>
<td>CorrectSteg IncorrectSteg CorrectNonSteg IncorrectNonSteg Don't Know</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>84</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E10_Color_nonMix_Standard_NonSorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Type: (Seen Images)</td>
</tr>
<tr>
<td>CorrectSteg IncorrectSteg CorrectNonSteg IncorrectNonSteg Don't Know</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>98</td>
</tr>
<tr>
<td>Test Type: (Unseen Images)</td>
</tr>
<tr>
<td>CorrectSteg IncorrectSteg CorrectNonSteg IncorrectNonSteg Don't Know</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>94</td>
</tr>
</tbody>
</table>

Figure 6.6 Experiments on standard FANN without sorting luminance

Figure 6.6 and Figure 6.7 show the test results on color images after training the FANN without sorting luminance values. From our experiments with color images, the shortcut type FANN was classifying the images better than the standard type FANN. In the case of the standard type FANN, most of the images were classified into the Don’t Know section. This suggests that FANN is not training well with this data set. We also conjecture that the randomness of training data affects the test results, at least on small data sets. Future testing with much larger data sets may provide more insight, although we suspect this won’t be the case. While luminance changes can be detected statistically, it is possible that we will not be able to use this characteristic to successfully train an artificial neural network. The idea for conducting experiments with sorting luminance was based on the statistical experiments. It was one of a number of variables we tested.
with to see what the results would turn out to be. We are surprised at the difference in sorting the data versus not sorting it, but do not yet have a sound explanation for this.

Figure 6.7 Experiments on shortcut FANN without sorting luminance

6.2.4 Test Runs on Steganalysis Tools

In order to start a basis for comparison of the effectiveness of our ANN steganalysis model, we ran tests using StegSecret and StegSpy, the freely available steganalysis tools discussed in Chapter 3. The steg images used to train and test our ANNs are used in these experiments as well.

6.2.4.1 StegSecret

StegSecret allows us to check for the presence of steganography in a collection of images. All 250 color and 250 grayscale steg images used in the FANN experiments were given to the StegSecret tool to run for analysis. Unfortunately, StegSecret was not able to
detect the presence of the steg in any of the images tested. We note that StegSecret was selected because it claimed to detect LSB type steganography.

6.2.4.2 StegSpy

StegSpy is a difficult tool compared to StegSecret, where each image must be entered one by one, and then waits for the analysis before testing another image.

<table>
<thead>
<tr>
<th>CorrectSteg</th>
<th>IncorrectSteg</th>
<th>CorrectNonSteg</th>
<th>IncorrectNonSteg</th>
<th>Success%</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>42</td>
<td>37</td>
<td>13</td>
<td>45</td>
</tr>
</tbody>
</table>

We selected a collection of 50 color steg images made from 50 clean color images from the larger data set used in the FANN experiments. Table 6.1 shows the results for the images given to the StegSpy. This tool does not give good results, particularly in light of the 42 images with data embedded in them that were completely missed by this analysis. Testing with 100 grayscale images consist of 50 clean and 50 steg, this tool could not correctly identify a single steg image.
CHAPTER VII
CONCLUSION AND FUTURE WORK

Our goal of this research is to investigate the viability of training an artificial neural network to detect the presence of steganography. In this initial research, we can see that the shortcut type neural network does make a positive difference compared to the standard artificial neural network. In case of standard type, FANN was giving same response for both steg and non-steg images; unlike that of shortcut type.

Our preliminary experimental results show that Hide4PGP can be detected to some extent by training with the shortcut artificial neural network. More experiments need to be performed on images that are embedded using steganographic tools other than Hide4PGP. We hope to be able to extract unique steg patterns that can distinguish the steg images from non-steg images. We conjecture that studying and training the FANN with the steg footprints will lead to more positive results.

For detecting image steganography, we have tested using pixel luminance of an image for distinguishing steg and non-steg images. More study needs to be conducted to extract the image properties that could be changed while conducting the process of steganography. These image properties can then be used for distinguishing steg images from non-steg images.
For future work, more experiments should be conducted in order to determine the number of hidden layers which is most appropriate for steganographic detection. The network may not able to learn well if we use too few layers. Also a thrashing situation can occur if too many layers are used, because the network cannot clearly identify any unique properties. Another factor that determines efficiency is the number of nodes in each of the hidden layers. Since the numbers of nodes in the input and output layers must be constant, the number of nodes in the hidden layer has to be determined by experimental runs. These factors also play a crucial role in network training time.

We also plan to train the FANN by feeding only the least significant bit of the images to the ANN and see how well it performs. The rationale for this is there are so many bits that are the same, they convolute the data fed in, meaning, the real data that is different, gets overshadowed by the identical bits. This assumes the case where images have been modified using a LSB algorithm.

In our research, we have focused on image steganalysis. The research should be extended to detect other types of steg carriers such as text, audio and video. If we are able to extract the steg properties from these carries, then the FANN can be trained for these other types as well. We should also take into consideration other types of artificial neural network software than FANN. This research is only the beginning phase of an investigation into the viability of using artificial neural networks to detect the presence of steganography.
REFERENCES


[34] Jason R.Bowing, Priscilla Hope, Kathy J.Liszka; Spam Image Identification Using an Artificial Neural Network, MIT SPAM Conference 2008.


[36] Pcmag, http://www.pcmag.com/encyclopedia_term/0,2542,t=luminance&i=46393,00.asp


[38] Neil F. Johnson and Sushil Jajodia; Steganalysis of Images Created Using Current Steganography Software, pp 281-283